Object Detection Walk-through

NB We are using tensorflow 1.13 for this module, tensorflow 2.0 beta is still on the test phase. Most model available are still on tensorflow 1_version some of the coding is being phased out. So until they update the models to 2.0 you rather stick to the highest version of version 1.

You can also run this system on your local machine without anaconda using other python GUI

1. **Set_up**

I assume you have installed anaconda or follow the following link

https://problemsolvingwithpython.com/01-Orientation/01.03-Installing-Anaconda-on-Windows/

Create new environment by conda

If you are unwilling to create conda environment (maybe because of lazy), you can skip this section. However, I strongly recommend you to create this **for the convenience in the future**.

Run the command below:

```
conda create -n tf
```

If you do not have tensorflow installed

First, you need to change to the env you have just built by conda:

```
conda activate tf
```

Afterwards, type in the command to install TensorFlow preferably version 1.13.1, you need:

```
conda install tensorflow-gpu
```

It will install the latest version, so you will need to specify the version.

If you want to install a specific version of tensorflow-gpu or cpu veison, you can change the command like this:

```
conda install tensorflow-gpu=1.13.1  #if you want to install 1.13.1 version
conda install tensorflow  #if you want to install cpu version
```

After anaconda solve the environment, you just need to type in 'y' to confirm the installation.

Anaconda will automatically install other libs and toolkits needed by tensorflow(e.g. CUDA, and cuDNN), so you have no need to worry about this.

Type in `python` to enter the python environment.

```
import tensorflow as tf
tf.__version__
```

When you see the version of tensorflow, such as 1.13.1, you have successfully install it.

That's all, Thank you.

Backdate to TensorFlow 1.13.1

With the current version of TensorFlow (2.0) there are some incompatibilities between TensorFlow 1.13.1 and models (more on models later) 2.0. To Remedy these incompatibilities it is necessary to backdate TensorFlow to 1.13.1.

Check your TensorFlow version:

```
pip list
```

```
If you have not installed pip,
```

```
Conda install pip
```

If not 1.13.1, open an ***administrative*** command prompt and enter:

```
pip uninstall tensorflow
```
(wait until the uninstall is complete)

(if using TensorFlow CPU version)
```
pip install --upgrade tensorflow==1.13.1
```

(if using TensorFlow GPU version)
```
pip install --upgrade tensorflow-gpu==1.13.1
```

2. Clone the repository containing this document

To clone the repository install git on windows

https://www.computerhope.com/issues/ch001927.htm

To clone

https://git-scm.com/docs/git-clone

Clone the repository containing this document:

In git:

cd “C:\Users\131324\Documents\ML”

git clone
https://github.com/Mutekeri/Tensorflow-Object_detection-on-windows-machine.git

to a location you’d like to work in, for example I’ll use:

C:\Users\131324\Documents\ML

You can choose any directory you’d like. Going forward in this document we’ll refer to this location as:
(repository_location)

3. Follow TensorFlow Object Detection API readme installation instructions

Google on “TensorFlow Object Detection API”, this should take you to:

https://github.com/tensorflow/models/tree/master/research/object_detection

Scroll down to the readme.md and, under "Setup", choose the "Installation" link, this should take you to:

https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md

In subsequent steps we will more or less follow these Object Detection API instructions, will some modifications due to these instructions being intended for Mac / Linux use and this document being a Windows walk-through.

Open an ***administrative*** command prompt, then copy/paste in the following command (modified for Windows use):

```
pip install pillow lxml jupyter matplotlib pandas
```

4. Clone the TensorFlow "models" repository

Create a directory "C:\TensorFlow" (or use a different drive letter if you prefer), then clone the TensorFlow "models" repository:
https://github.com/tensorflow/models
to this location.

```
cd C:\TensorFlow\models
```

5. Download and Compile Protobuf

Continuing with the TensorFlow Object Detection API readme.md installation instructions, Google on "Google Protobuf GitHub", this should take you to https://github.com/google/protobuf
Choose "releases" (top-right center-ish on the page) and download the Windows version of either the latest release, or v3.4.0 if the latest release results in errors in the subsequent steps. At the time of this writing I've found v3.5.1 cannot successfully parse the *.proto command below so for the moment version v3.4.0 is recommended. If when you're reading these instructions a substantially newer version is available you may want to try the newer version.

In any case, download the applicable win32.zip, ex. "protoc-3.4.0-win32.zip" or "protoc-X.X.X-win32.zip" if you're trying a more recent version. Bear in mind that you will need to adjust the rest of the commands in this document per your chosen protoc version.

Extract your download and copy the extracted directory to C:\, so as to end up with Protobuf at the location "C:\protoc-3.4.0-win32". Note that this directory should include "bin\", "include\", and "readme.txt"

Navigate to "C:\TensorFlow\models\research\object_detection\protos", you will find many files ending in .proto, but none ending in .ipynb. The next step will create .ipynb files in this directory.

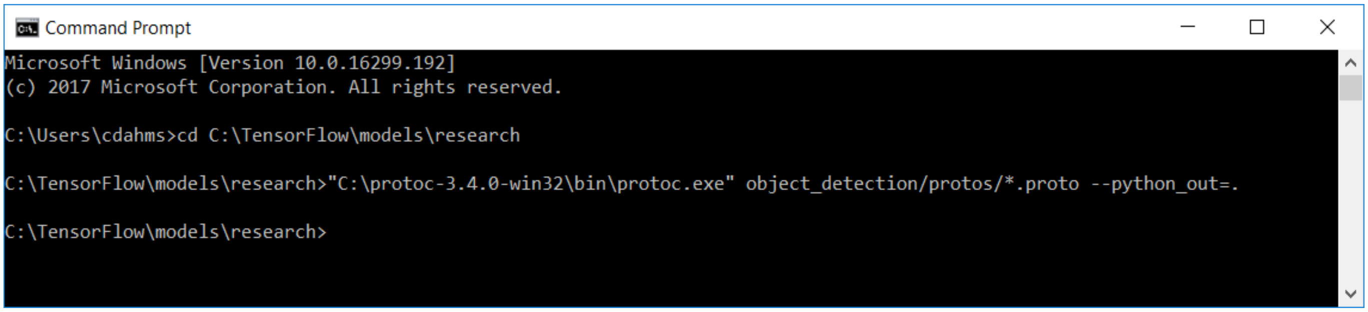
Bring up a command prompt and cd into the models\research directory with the following command:

```
cd C:\TensorFlow\models\research
```

then enter the following command to compile protoc:

```
"C:\protoc-3.4.0-win32\bin\protoc.exe" object_detection/protos/*.proto --python_out=.
```

If this command works, there will be no message, like so:



If this command does not work, there will most likely be an error message.

Note that you have to either enter the protoc-3.4.0-win32 directory in your PATH, or specify the full path at the command prompt as show here.

Next navigate back to ***C:\TensorFlow\models\research\object_detection\protos***, if the compile was successful, there will now be a .ipynb file for each .proto file.

6. Test the TensorFlow Object Detection API using the pre-trained model within a Jupyter Notebook

Open a command prompt and cd to the \models\research\object_detection direction, ex:

```
cd C:\TensorFlow\models\research\object_detection
```

Note that this directory contains a "object_detection_tutorial.ipynb" Jupyter Notebook file. Enter this command at the command prompt to start this notebook

```
jupyter notebook
```

This should start a Jupyter Notebook in your default browser.

In the Jupyter Notebook in your browser, choose "object_detection_tutorial.ipynb", then at the top choose "Cell", then "Run All".

The notebook script will take perhaps a few minutes to run, unfortunately Jupyter Notebooks do not seem to provide any kind of progress bar or indication the script is running. After a few minutes (perhaps slightly longer depending on your computer and internet connection speed) you will see two example output images at the bottom of the page, one of two dogs being identified, and the other of various people and kites being identified at a beach.

The actual detection is pretty quick, most of the time in this example is downloading the pre-trained COCO model (more details on this later).

It might not run, depending on the set up of your anaconda path

Copy the `object_detection folder to Tensorflow-Object_detection-on-windows-machine folder`
Then zip the folder using compression in windows to make it a zip file

Open a new folder in jupyter call it what you want
Open a new python file
Type in

```
import zipfile as zf
files = zf.ZipFile(C:\Users\131324\Documents\ML\Tensorflow-Object_detection-on-windows-machine.zip,'r')
files.extractall('object_detection')
files.close()
```

Open the object detection folder, you will find another folder older
Go back
Rename the object detection to object_detetion1 by adding a 1
Go inside the folder, move the other object_detection folder just outside by removing /object_detetion1 on the path, this is to prevent jupyter notebook to have errors when finding the information in your file.
There will be two folders object_detection1 and object_detection, delet the object_detection1 and open again
You will see the relevant files straight up, without getting in another folder
In the Jupyter Notebook in your browser, choose "object_detection_tutorial.ipynb", then at the top choose "Cell", then "Run All" or run step by step. This will show you pictures

You move the "Detect Objects using web camera. ipynb" to the object_detection folder. then at the top choose "Cell", then "Run All" or run step by step. It will activate your web camera, detecting objects.

7. Run *Step_1_Verify models.ipynb* to test the TensorFlow Object Detection API using the pre-trained model

In ***(repository_location)*** open the file "Step_1_Verify models.ipynb" in your chosen Python editor, ex. PyCharm. You may notice red underlines beneath the following two lines at the top:

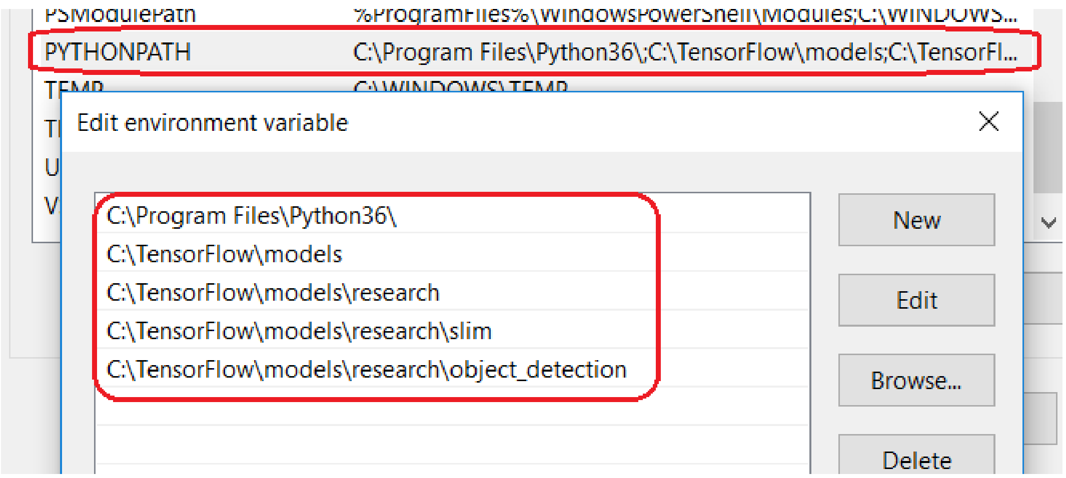
```python
from utils import label_map_util
from utils import visualization_utils as vis_util
```

If you attempt to run the script you will likely get an error to the effect of "ModuleNotFoundError: No module named 'utils'", this is because we haven't yet informed Windows of the location of the TensorFlow models repository so the script can't find the necessary libraries when ran. To remedy this, proceed as follows:

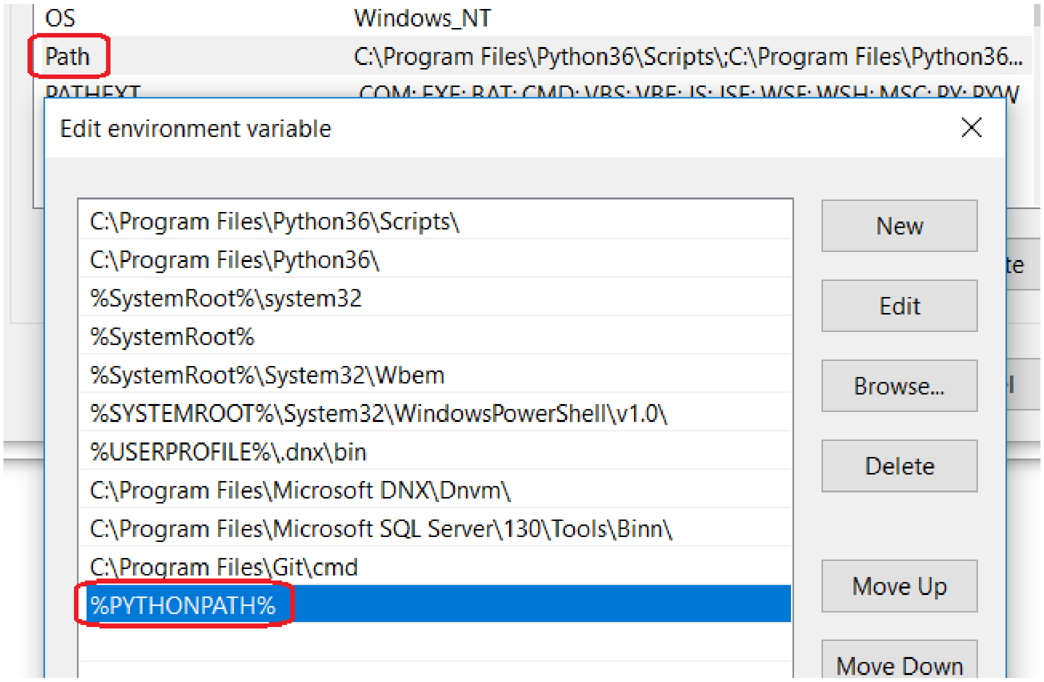
Go to System -> Advanced system settings -> Environment Variables -> System Variables -> New, and add a variable with the name PYTHONPATH and these values:

```
C:\Program Files\Python36\
C:\TensorFlow\models
C:\TensorFlow\models\research
C:\TensorFlow\models\research\slim
C:\TensorFlow\models\research\object_detection
```

When you're done PYTHONPATH should look like this:

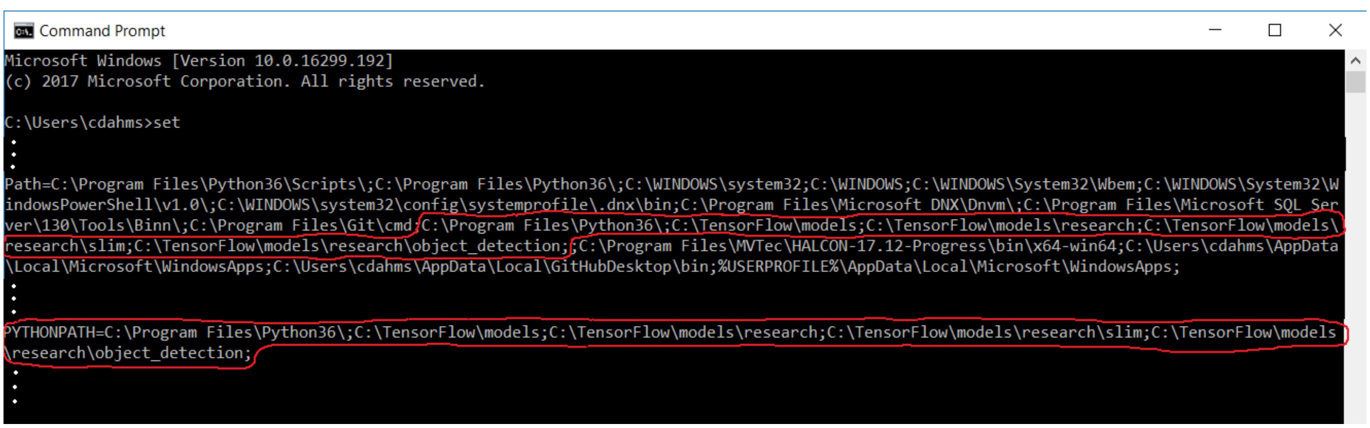


Next, while still in Environment Variables, edit PATH and add %PYTHONPATH% like so:



Reboot so the path changes take effect (do NOT skip this step)

After the reboot is complete, pull up a command prompt and type “set” with no parameters, this will show the contents of all your environment variables. Verify PATH and PYTHONPATH have the values you entered above:



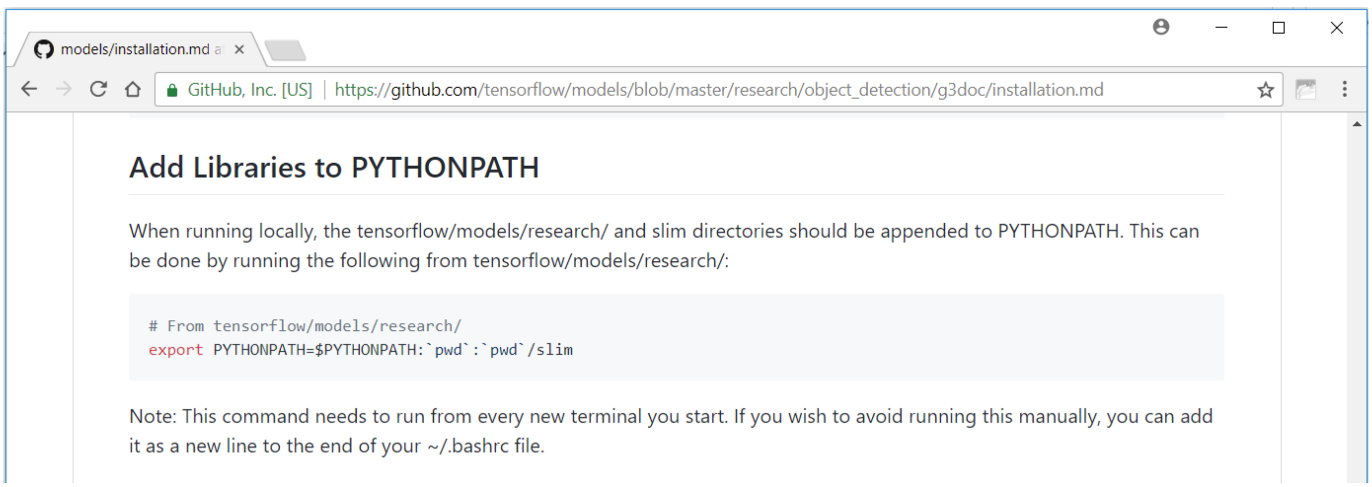
In ***(repository_location)*** open the file “Step_1_Verify models.ipynb” again, the red underlines under the from utils import lines should now be gone. Verify the paths in the “module level variables” section at the top are correct, then go ahead and run the script. It should run successfully, producing substantially the same result as the object_detection_tutorial.ipynb file did in the notebook, the only difference being the two images will be shown one at a time in OpenCV windows. Click on the image window and press any key to go to the next image or end the program if on the 2nd image.

The above settings are for those running without anaconda, the where you state the location of your python file. You can still follow the settings but make sure it recognize your anaconda python location. You can type in set in both cmd or anaconda to check if there have the same files.

Once this script runs from a directory other than C:\TensorFlow\models\research\object_detection you've proven out the path entries above and should now be able to run this script, or any script that uses `from utils import` from any directory.

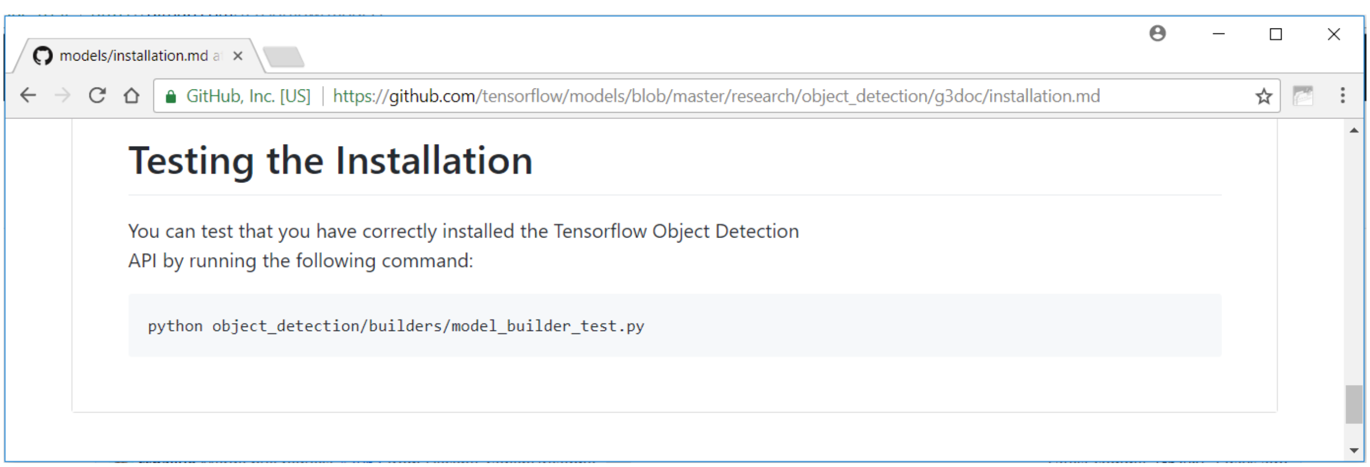
Take a moment to skim the "Step_1_Verify models.ipynb" code, you don't have to understand the details of every line fully but try to familiarize yourself with the gist of the steps involved. You should be able to verify the content of this file is similar to the "object_detection_tutorial.ipynb" contents, with relatively minor modifications to adjust from being ran as a regular Python file rather than within a Jupyter Notebook.

Note that the TensorFlow Object Detection API Installation instructions mentioned above state the following:



The PYTHONPATH and PATH editing steps we just performed are the Windows equivalent of this.

Once we've gotten "Step_1_Verify models.ipynb" to run the TensorFlow Object Detection API install is complete. If you'd like to complete the TensorFlow Object Detection API installation instructions, the final step as stated is:



Since we've verified "Step_1_Verify models.ipynb" runs this isn't really necessary at this point, however if you'd like you can copy the file
C:\TensorFlow\models\research\object_detection\builders\model_builder_test.py
to any directory and verify it runs as well, the output of running this script should be similar to:
....

You have to copy the model_builder_test.py into the jupyter notebook by opening a blank file and copy it. You can break it down documenting and running it.

Ran 11 tests in 0.089s

OK

Now that we've gotten Step_1_Verify models.ipynb to run using the pretrained model, we'll make our own model

8. Download Images for training / testing

Google has now datasets for AI
Google data sets and explore
https://toolbox.google.com/datasetsearch
https://ai.google/tools/datasets/

Download at least 110 images (preferably more) of an object you'd like to identify in a scene. You can use my downloaded images of scenes with traffic lights if you'd like to save time on this step:

Google Drive:
https://drive.google.com/drive/folders/1pWnmPuMv9LJ5Cw0UybVaHJGXTJ9QECRF?usp=sharing
OneDrive:
https://1drv.ms/f/s!AoYpNs_C1pusgxm5ziwTZtL6B7Io

Thanx to github.com (MicrocontrollersAndMore) for providing his personal drives for practice data on Google Drive and my OneDrive with these images so if one or the other ever goes missing the other will still be available. The images are the same on both.

Downloading at least 110 images is recommended so when we separate out 10 images for testing, we will still have at least 100 different images for training.

9. Use the program "labelImg" to create a .xml file to go with each scene image

Again github.com (MicrocontrollersAndMore) provides the data, If you'd like to save time for the moment on this step, you can download images with the already created labels:

Google Drive:
https://drive.google.com/drive/folders/1OFx1QpVFx7A0JZA4GTgB4Rm1lIFDemlm?usp=sharing
OneDrive:
https://1drv.ms/f/s!AoYpNs_C1pusgxo-0saTbmrEUw1N

However, I suggest going through the labeling process. It's not difficult and you will need to know how to do this when you use your own images later on.

Google on "labelImg GitHub", that should take you to https://github.com/tzutalin/labelImg

In the readme.md, scroll down to the "Download prebuild binaries" section, then choose the "Windows & Linux" link. Scroll to the bottom of the next screen and click on the most recent Windows version download, ex. "Windows_v1.6.0" as of this writing.

Make a directory in Program Files named labelImg_vX.X.X where X.X.X is your version number, ex:

C:\Program Files\labelImg_v1.6.0

Then extract the labelImg pre-build binary zip to this location. If the zip contains an extra layer of folder with the same name as the zip you can remove the unnecessary layer, the idea is the contents of the zip, currently an executable "labelImg.exe" and a directory "data" should end up in "C:\Program Files\labelImg_v1.5.2".

Double-click on labelImg.exe to start the program, it should start and run without any further steps. You may want to add a desktop shortcut and/or pin to the start menu or taskbar for convenience.

To use labelImg, choose "Open Dir" on the left, then choose the directory with your saved images in it. "File List" at the bottom right will list all the images in the directory, and you can use the "Next Image" and "Previous Image" buttons to navigate back and forth. Note that files are navigated through within labelImg in ASCII order not numeric order, i.e. the order of 1.jpg, 2.jpg, 10.jpg, and 100.jpg would be 1.jpg, 10.jpg, 100.jpg, and 2.jpg.

Use Ctrl + mouse wheel to zoom out or in. Note that when zooming in the window will zoom in towards where the mouse cursor is, so for example if you'd like to zoom in toward the top left of the image, put the mouse cursor towards the top left, then use Ctrl + mouse wheel to zoom in. Once zoomed in there is not currently a pan feature, however you can use the horizontal and vertical scroll bars to move around.

When you're ready to draw a box around an object you'd like to train on, simply choose "Create RectBox" on the left and left-click and drag a box with the mouse. For the first box drawn, you will be prompted to enter a text description of the classification, for example, "traffic_light". Make sure to not include a space in the name, use underscores if necessary. If you enter the classification description in the "Use default label" text box and check the adjacent check box then you won't be prompted for this each time.

When you've drawn a box around each instance of an object you'd like to train on in an image, choose "Save" on the left and labelImg will prompt to save an .xml file in the same directory as the image with the same name, but ending in .xml instead of an image file extension.

Once you get the hand of the labelImg UI you'll find that it only takes perhaps 5-10 seconds to label each image, so you can label quite a few images in a relatively short amount of time.

Take a look at you're xml files as you go, they should look something like this:

```
<annotation>
	<folder>traffic_lights_with_info</folder>
	<filename>1.jpg</filename>
	<path>/home/cdahms/Dropbox/OpenCV pics and stuff/traffic_lights_with_info/1.jpg</path>
	<source>
		<database>Unknown</database>
	</source>
	<size>
		<width>1259</width>
		<height>942</height>
		<depth>3</depth>
	</size>
	<segmented>0</segmented>
	<object>
		<name>traffic_light</name>
		<pose>Unspecified</pose>
		<truncated>0</truncated>
		<difficult>0</difficult>
		<bndbox>
			<xmin>441</xmin>
			<ymin>321</ymin>
			<xmax>472</xmax>
			<ymax>383</ymax>
		</bndbox>
	</object>
	<object>
		<name>traffic_light</name>
		<pose>Unspecified</pose>
		<truncated>0</truncated>
		<difficult>0</difficult>
		<bndbox>
			<xmin>990</xmin>
			<ymin>267</ymin>
			<xmax>1028</xmax>
			<ymax>348</ymax>
		</bndbox>
	</object>
</annotation>
```

Hopefully labelImg will be around for a long time as it's fantastic for this purpose, but just in case something unusual should happen to the original I'll keep the fork on my GitHub as a back-up.

Before we move on to the next step, some anti-virus programs seem to flag labelImg as a virus:

https://www.virustotal.com/#/file/7111532720254e72f53441b0e9854eb851b2d9a6871e11568fb3cfce28fba7f8/detection

There is an issue filed on the labelImg GitHub pertaining to this:

https://github.com/tzutalin/labelImg/issues/178

As of version 1.6.0, Windows Defender does not report labelImg to be a virus. I've built labelImg from source and used it under Ubuntu 16.04 and had no problems. Somebody who is very skeptical may point out that, when using a pre-built binary, it is possible the author could have put malware in the pre-built binary that is not in the repository source on GitHub, however I feel this is extraordinarily unlikely. labelImg has a substantial user community, if the pre-built binary contained malware surely this would have been mentioned in the repo issues or on an online forum somewhere (I was not able to find any comments this effect anywhere).

If you are obsessed over security, you could read every line in the labelImg GitHub repository to assure there is not any malicious content, then follow the readme.md instructions to build from source. However, I've used the pre-build binary for a while with no ill effects, I'm pretty confident it's more than safe and reviewing every line of source then compiling from source is almost for certain not necessary. However, I cannot personally guarantee this so if you demand the utmost in security then reviewing every line of code and compiling from source is the only option.

10. Make *training_images* and *test_images* directories, then copy each image / .xml pair into these

In ***(repository_location)***, make a directory "training_images" and a directory "test_images". From wherever you saved your images and associated .xml files from the previous step, copy (make sure to copy rather than move to preserve the originals) them all into the "test_images" directory you just made, for example in my case I would copy the images and the associated .xml files to:

```
\training_images
\test_images

There will all occur in the folder with the scripts.
```

Note that although the file location is included in the .xml files, this will be removed when we convert to .csv in the next step, so it is not necessary to update this file location when moving the .xml files.

Next, choose 10 images and the associated .xml files to test on and move (don't copy) those from ***(repository_location)***\training_images into ***(repository_location)***\test_images. Note that we are separating out the images we will use before training (the next step) so the images we test on will not have been used for training. You can do it in your local file, and zip it using the zip steps above.

11. Run *Step_2_xml_to_csv.ipynb* to convert the .xml file for each image to 2 .csv files, one for training and one for evaluation

Next, in ***(repository_location)***, make a directory “training_data”, the script we are about to run will write two .csv files which summarize the .xml content to this location.

With your chosen Python editor, in ***(repository_location)*** open the file “1_xml_to_csv.ipynb”. Verify the paths in the module level variables section at the top are correct, the run the script, then check the data directory in ***(repository_location)*** and you should find “train_labels.csv” and “eval_labels.csv”, for example in my case in location of these files would be:

In the folder where the ipynb file you ran is located.

training_data
```
\data\train_labels.csv
\data\eval_labels.csv
```

Open these files in Notepad or any other editor that will show .csv file content without suggesting to convert to a different format. Opening these .csv files with Excel is not recommended since Excel will try to get you to convert to .xlsx file format which we don’t want to do. Verify the files are not empty and contain roughly the correct number of lines for the images you used. Note that there will be a line in each .csv file for each box drawn, not a line for each image. i.e. if in your training set you had 100 images and drew a total of 250 boxes, there should be 250 lines in your train_labels.csv file (not 100 lines).

12. Run *Step_3_Generate_tf_records.ipynb* to Create training and testing .tfrecord files

Using your chosen Python editor, in ***(repository_location)*** open the script “Step_3_Generate_tf_records.ipynb”. In the module-level variables section at the top, verify the paths are correct.

Next, scroll down to the “class_text_to_int” function and update the if-else statement per the classification names you used when generating the .xml files to go with each image above. For example, if you’re using only one classification called “traffic_light” in my example, your if statement should be:

```python
if row_label == 'traffic_light':
    return 1
else:
    print("error in class_text_to_int(), row_label could not be identified")
    return -1
# end if
```

If you’re using 3 classifications, “traffic_light”, “stop_sign”, and “yield_sign”, your if statement would be:

```python
if row_label == 'traffic_light':
    return 1
elif row_label == 'stop_sign':
```

```
    return 2
elif row_label == 'yield_sign':
    return 3
else:
    print("error in class_text_to_int(), row_label could not be identified")
    return -1
# end if
```

Next, run the script and verify no errors occur, then check your ***(repository_location)***\data directory, you should now find 2 more files, “train.tfrecord” and “test.tfrecord” in training data.

13. Choose, download, and extract a model (“ssd_inception_v2_coco” is recommended)

Google on “TensorFlow GitHub”, that should take you to https://github.com/tensorflow. Next, go to the “models” repository, then research -> object_detection -> g3doc -> detection_model_zoo.md, this should take you to
https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

This page lists the available models. You can also train your own model, but that is very time consuming and in most cases not necessary as there are quite a few stock models to choose from.

You can do some Googling to read up further on the models available. When choosing a model there is inherently a speed / accuracy tradeoff to some degree, also some models are better at detecting certain types of objects than others.

To make a long story short, probably the best all-around choice for detecting most objects in most images is “ssd_inception_v2_coco”, which is the model we will choose in our example here.

For other potential choices, if you need the fastest detection possible (i.e. if you’re working with images being streamed from a webcam) you should probably choose “ssd_mobilenet_v1_coco”, which is intended to be fast and therefore able to run on mobile devices that do not have the full computing power of a desktop, hence the word “mobile” being in the name.

Inception provides very good accuracy, but if you need the highest accuracy possible at the expense of speed you can experiment with some of the other slower models that may provide moderately better accuracy in some cases. It will download the latest coco file in my case it was version 18 which still works.

Click on your chosen model and download it to your ***(repository_location)*** directory. This will be a .tar.gz file, ex. ssd_inception_v2_coco_2017_11_17.tar.gz.

Note that the model name includes a date which will change as Google updates the model over time. Take care to adjust the subsequent steps per the date in the model name in comparison to the date used in the video and this document if by the time you are following along the date has changed slightly.

Right click on the .tar.gz file and choose 7-Zip -> Extract Here, this will extract the .tar.gz file in to a .tar file, i.e. in ***(repository_location)*** you will now have "ssd_mobilenet_v1_coco_2017_11_17.tar.gz" and "ssd_mobilenet_v1_coco_2017_11_17.tar"

Right click on the .tar file (not the tar.gz file) and choose 7-Zip -> Extract Here, this will create a directory of the same name, except without the .tar at the end, i.e. you will now have a directory ***(repository_location)***\ ssd_mobilenet_v1_coco_2017_11_17. Navigate into this directory and verify it's not empty.

Once you have the file,
Zip it and use the zip steps to import it to the folder using the script.

14. Check / edit "label_map.pbtxt"

Open ***(repository_location)***\label_map.pbtxt in Notepad or a similar text editor and verify the contents are:

```
item {
  id: 1
  name: 'traffic_light'
}
```

If you're detecting something other than traffic lights, change the name to match the name you used when generating the .xml files with labelImg. If you're using multiple labels, add a { } section for each label type, with an incrementing id matching the other uses throughout this document.

For more examples of label map files for TensorFlow object detection you can view Google's examples here:
https://github.com/tensorflow/models/tree/master/research/object_detection/data

When complete, save your changes to label_map.pbtxt and close it.

15. Check / edit "ssd_inception_v2_coco.config"

Open ***(repository_location)***\label_map.pbtxt in Notepad or a similar text editor and skim the contents.
"ssd_inception_v2_coco.config" is a large configuration file that will be used by "3_train.ipynb" in the next step.

This file is from
https://github.com/tensorflow/models/blob/master/research/object_detection/samples/configs/ssd_inception_v2_coco.config
with some modifications.

If you are using a model other than Inception v2 then of course you will need to use a different config file from https://github.com/tensorflow/models/tree/master/research/object_detection/samples/configs and make the applicable modifications. Comparing ***(repository_location)***\label_map.pbtxt to

https://github.com/tensorflow/models/blob/master/research/object_detection/samples/configs/ssd_inception_v2_coco.config would probably be the best way to determine what modifications to make to any of the other config files located at
https://github.com/tensorflow/models/tree/master/research/object_detection/samples/configs.

The following settings are especially noteworthy:

num_classes: Change this to your number of classes if applicable. For example, if you're detecting traffic lights only, leave this as "1". If you're detecting traffic lights, stop signs, and yield signs, change this to "3".

num_steps: Google's default for this setting is 200000, which I've changed to 500 for a relatively quick training time, perhaps five hours on a moderate computer. Once you get this process to work all the way through, it would be recommended to re-train with a much higher value, ex. 5000, for production-grade results.

fine_tune_checkpoint, **train_input_reader input_path** and **label_map_path**, and **eval_input_reader input_path** and **label_map_path** are all set to relative paths currently. Verify these paths exist on your computer. If you experience "file not found" errors when running "3_train.ipynb" in the next step, try changing the paths to absolute paths. For example if you encounter an error similar to "fine_tune_checkpoint not found", try changing:
`fine_tune_checkpoint: "ssd_inception_v2_coco_2017_11_17/model.ckpt"`
to:
`fine_tune_checkpoint:`
`"C:/path/to/cloned/repo/location/…………./ssd_inception_v2_coco_2017_11_17/model.ckpt"`

Note that is seems best to use forward slashes in this file to separate files/directories, even though on Windows usually backslashes are used.

fine_tune_checkpoint: This is set as follows:
`fine_tune_checkpoint: "ssd_inception_v2_coco_2017_11_17/model.ckpt"`
This is a relative path to the model directory (which was downloaded as a previous step).

Note that there is not actually a file "model.ckpt", this actually refers to 3 files, "model.ckpt.data-00000-of-00001", "model.ckpt.index", and "model.ckpt.meta".

Of course if you are using an Inception model of a different version or a different date you will need to change the version and/or date to match the model you are using. If you are using a model other than Inception entirely, you will need to change this line completely, or better yet start with a different config file entirely from
https://github.com/tensorflow/models/tree/master/research/object_detection/samples/configs

train_input_reader: Currently these are set as follows, verify these (relative) paths/files exist:
```
input_path: "training_data/train.tfrecord"
label_map_path: "label_map.pbtxt"
```

eval_input_reader: Currently these are set as follows, verify these (relative) paths/files exist:
```
input_path: "training_data/eval.tfrecord"
```

label_map_path: "label_map.pbtxt"

When entering these paths, if you copy/paste this out of Windows File Explorer, take care to change the back slashes to forward slashes.

Additional things you may want to change include:

fixed_shape_resizer: the default width and height to resize your images to before training is 300 x 300, this is a good general value, but other values may work better for your images so you may want to try other values eventually

batch_size: Depending on your computer's memory you may want to lower this to something substantially lower than the default value of 24, especially if you experience memory errors in the subsequent steps. 10 may be a good all-around choice to use less memory but still produce a good result.

There are many other parameters that can be fine-tuned as well for better results in certain situations. Once you've completed this process successfully a few times through, skim through "ssd_inception_v2_coco.config" again and consider further experimenting with some of the other parameters for best results for a given training situation.

16. Run *Step_4_Train.ipynb* to perform the training

Run "Step_4_train.ipynb" either from the command line or from within your chosen Python editor. The necessary files and paths should already have been produced from the previous steps, if not, follow any error messages to resolve concerns as needed.

With the .config setting of num_steps set to 500 as described above this will take perhaps 3-4 hours to the whole night on a moderate computer, or substantially less if you have a high-end computer or have TensorFlow configured to use a GPU.

You might get an error

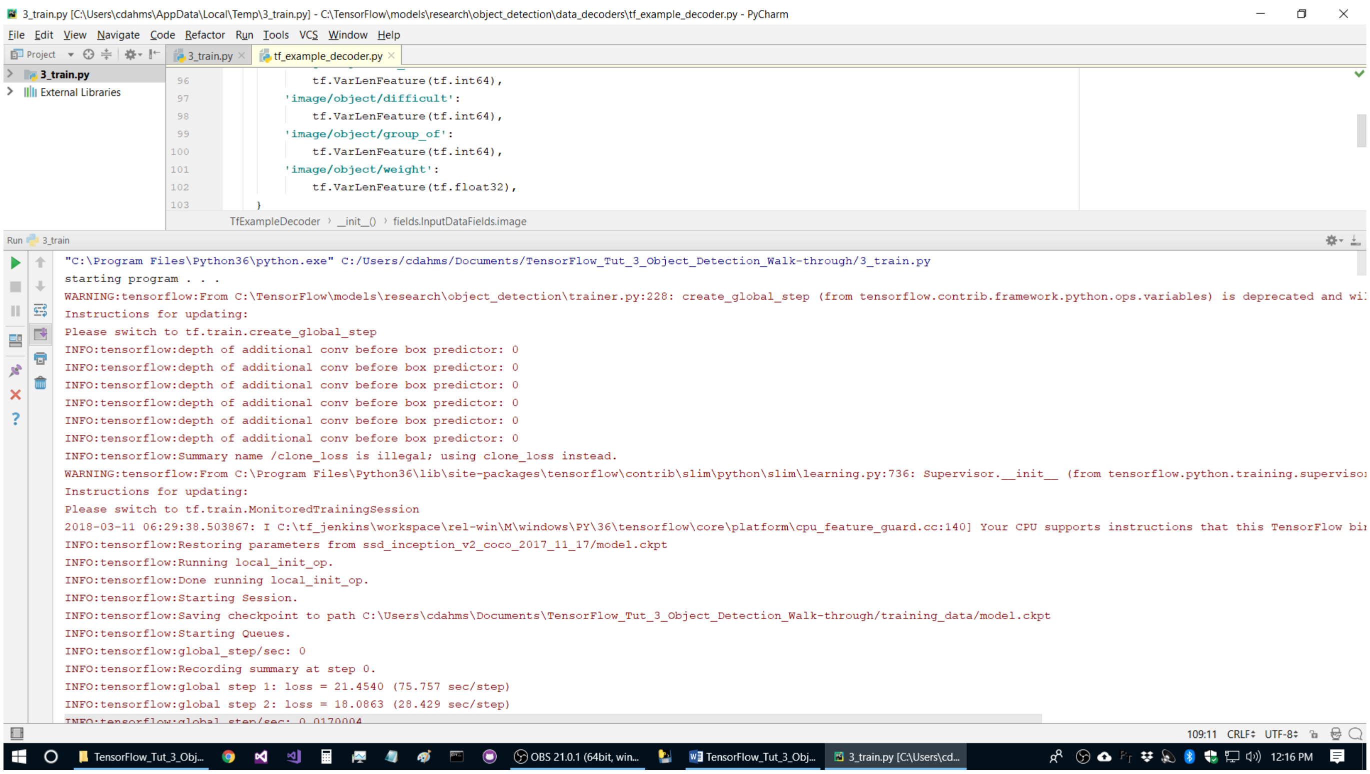
```
An exception has occurred, use %tb to see the full traceback.

SystemExit


C:\Users\Michael\Anaconda3\envs\tf\lib\site-packages\IPython\core\interactiveshell.py
:3334: UserWarning: To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

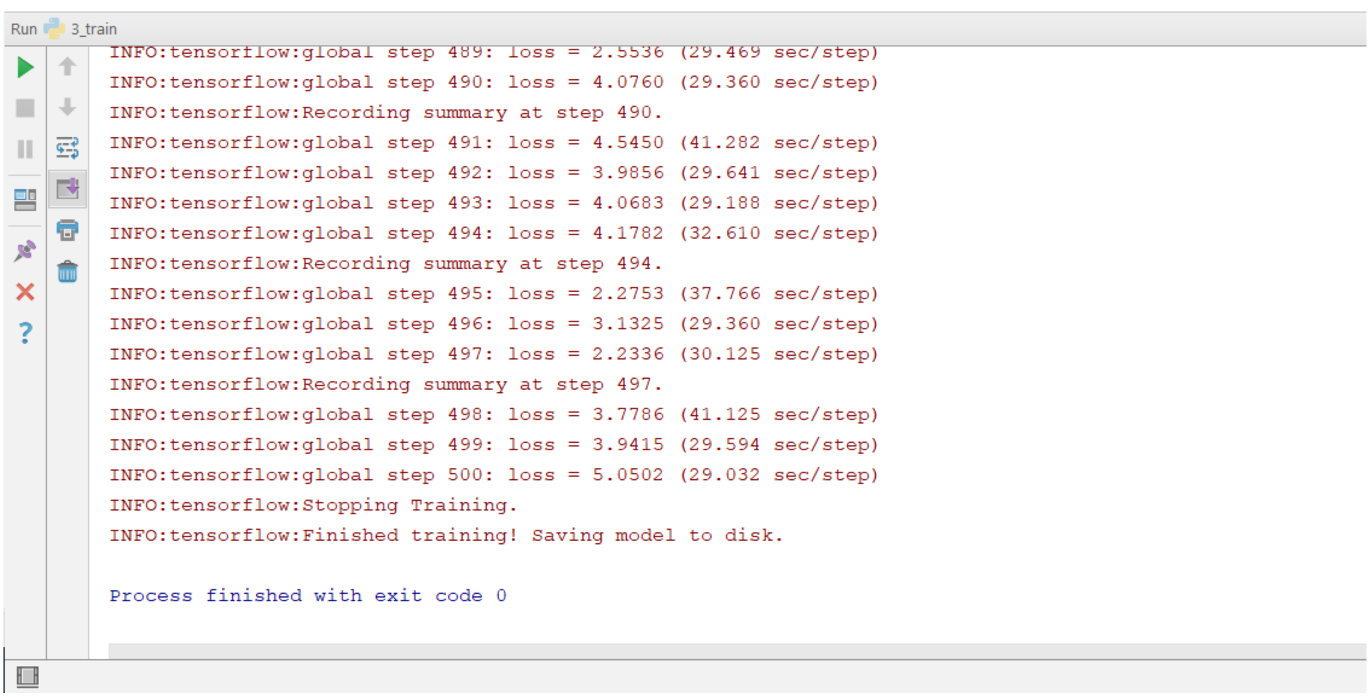
This is because the maximum has been reached of 500. It has worked, you might need to increase the training from 500, trial and error to see which one works better to accommodate the data.

The training should start out looking something like this:



As the training is proceeding, you will notice that files are written to in ***(repository_location)***\training_data.

If training completes successfully, you should see something similar to the following:



17. Run ***Step_5_Export_inference_graph.ipynb*** to export the Inference Graph

In your chosen Python editor open "Step_5_Export_inference_graph.ipynb" and give it a quick skim. Note that if you used a different number of steps than 500, you will have to change the "500" in TRAINED_CHECKPOINT_PREFIX_LOC to the number of steps you used.

Run "Step_5_Export_inference_graph.ipynb" to export the Inference Graph. This will take roughly a minute to run and will create a directory "***(repository_location)***/inference_graph"

If this process completes successfully you should see something similar to the following:

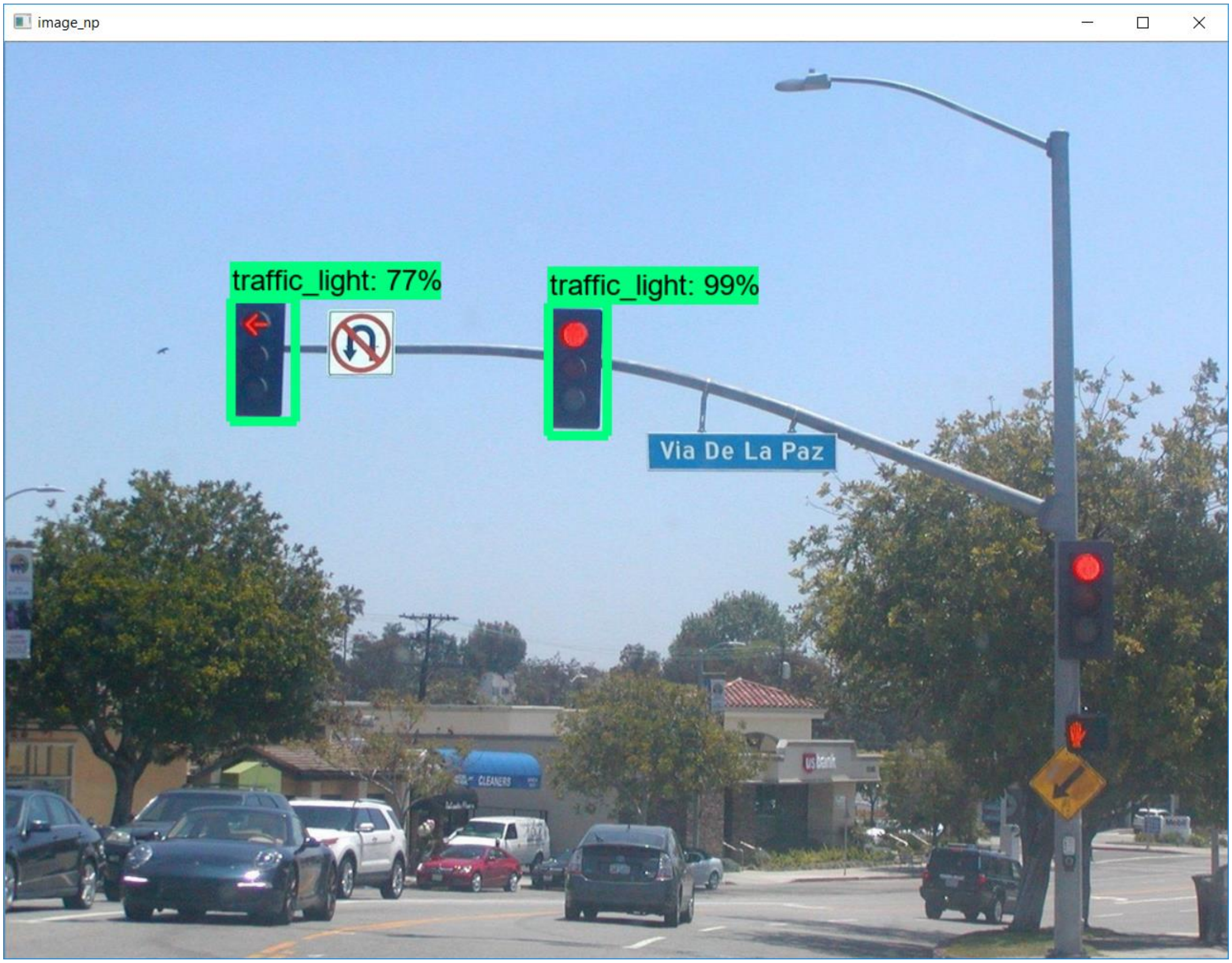
```
Command Prompt
C:\Users\cdahms\Documents\Tensor_Flow_Object_Detection_Walk-through>python export_inference_graph.py --input_type image_
tensor --pipeline_config_path "C:/Users/cdahms/Documents/Tensor_Flow_Object_Detection_Walk-through/ssd_inception_v2_coco
.config" --trained_checkpoint_prefix "C:/Users/cdahms/Documents/Tensor_Flow_Object_Detection_Walk-through/training/model
.ckpt-1000" --output_directory "C:/Users/cdahms/Documents/Tensor_Flow_Object_Detection_Walk-through/inference_graph"
WARNING:tensorflow:From C:\TensorFlow\models\research\object_detection\exporter.py:356: get_or_create_global_step (from
tensorflow.contrib.framework.python.ops.variables) is deprecated and will be removed in a future version.
Instructions for updating:
Please switch to tf.train.get_or_create_global_step
2018-01-22 04:44:45.776270: I C:\tf_jenkins\home\workspace\rel-win\M\windows\PY\36\tensorflow\core\platform\cpu_feature_
guard.cc:137] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX AVX2
2018-01-22 04:45:03.497682: I C:\tf_jenkins\home\workspace\rel-win\M\windows\PY\36\tensorflow\core\grappler\devices.cc:5
1] Number of eligible GPUs (core count >= 8): 0
Converted 410 variables to const ops.

C:\Users\cdahms\Documents\Tensor_Flow_Object_Detection_Walk-through>
```

Once this is complete, check the ***(repository_location)***\inference_graph directory and verify that checkpoint, frozen_inference_graph.pb, and model.ckpt.* files are there.

18. Run ***Step_6_Test*** to test on your own images

From ***(repository_location)***, open the file "5_test.ipynb" in your chosen Python editor. In the module level variables section at the top, verify the TEST_IMAGE_DIR properly points to your test images directory, then run the script. This script will show each of the images in your test images directory one by one, simply click on the image and press any key to go to the next image. Your results will hopefully look something like this:



Note that one of the 3 traffic lights is missing. Following the steps above, you will likely find you'll get a traffic light detection rate of 50%, or moderately higher.

The main reason this detection rate is not all that great is the relatively low training set size of about 100 images. For something production grade, 10,000+ images would be recommended. Also, upping the num_steps configuration parameter to 10000+ would likely improve accuracy.

Done!!

If you used my images your first time through, the next step is to perform the process again on your own images. For my next two tutorials, I'm planning on re-doing the content from tutorials 2 & 3 in C++ and .NET, I'm not sure in which order.